组成原理实验课程第_三_次实报告

实验名称	寄存器堆的实现			班级	李涛
学生姓名	艾明旭	学号	2111033	指导老师	董前琨
实验地点	A306		实验时间	2023.4.18	

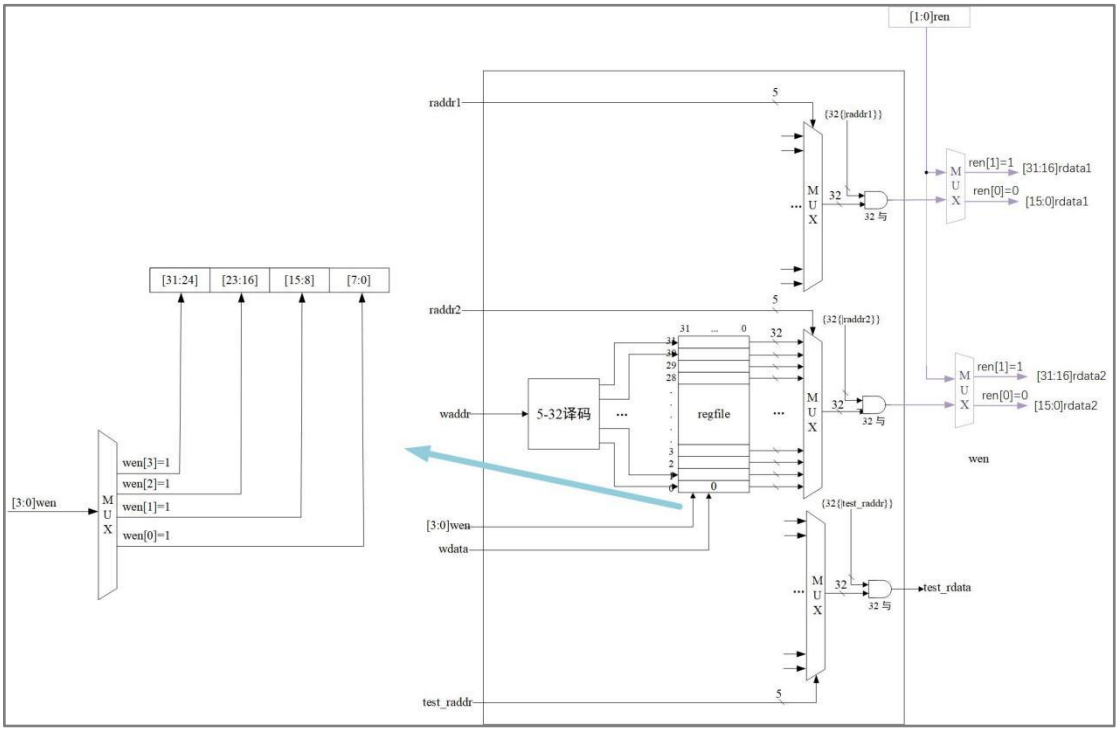
1、 实验目的

1. 熟悉并掌握 MIPS 计算机中寄存器堆的原理和设计方法。
2. 初步了解 MIPS 指令结构和源操作数/目的操作数的概念。
3. 熟悉并运用 verilog 语言进行电路设计。
4. 为后续设计 cpu 的实验打下基础

2、 实验内容说明

学习寄存器堆的概念，实验 verilog 语言在 ViVado 上实现寄存器堆并上实验箱实验；此外在原有寄存器堆代码的基础上，实现将输入数据按 BYTE 输入（原数据 32 位），并使用四个开关控制；读出数据指定寄存器高低 16 位的值，使用两个开关分别控制读取 read1 和 read2 的高低位。

3、 实验原理图



写入数据时，根据 wen 的四位输入决定将 wdata（8 位）写入指定寄存器，右上角是读出数据，根据 ren 的取值，来决定输出指定寄存器的高低位。

4、 实验步骤

（1）regfile.v 文件中将寄存器堆改变 wen 位 4 位，增加两位 ren 输入，并调整相关操作逻辑

I. 输入部分

wen 修改为 4 位，控制输入的位置；增加 ren 接口，控制输出的高低位；将写入 wdata 改为 8 位

II.

```
module regfile(
    input                   clk,
    input       [3: 0]wen,
    input       [1: 0]ren,
    input       [4 :0] raddr1,
    input       [4 :0] raddr2,
    input       [4 :0] waddr,
    input       [31:0] wdata,
    output reg [31:0] rdata1,
    output reg [31:0] rdata2,
    input       [4 :0] test_addr,
    output reg [31:0] test_data
    );
```

wen[2]输入次高 8 位、wen[3]输入最高 8 位

```
    always @(posedge clk)
    begin
        if (wen[0])
        begin
            rf[waddr][7:0] <= wdata[7:0];
        end
        if(wen[1])
        begin
            rf[waddr][15:8] <= wdata[15:8];
        end
      if(wen[2])
      begin
            rf[waddr][23:16] <= wdata[23:16];
      end
      if(wen[3])
      begin
            rf[waddr][31:24] <= wdata[31:24];
      end
    end
```

ren = 11 全部输出、　　ren = 10 输出高 16 位

ren = 01 输入低 16 位、 ren = 00 输出 32’d0

```
//读取 rdata1 高低位置
 always @(*)
 begin
        if (ren==2'b10)
            begin
            rdata1 <= rf[raddr1][31:16];
            end
         if (ren==2'b01)
            begin
            rdata1 <= rf[raddr1][15:0];
            end
          if (ren==2'b11)
            begin
            rdata1 <= rf[raddr1];
          end
          else
            begin
            rdata1 <= 32'd0;
            end
  end
```

```
//读取 rdata2 高低位置
    always @(*)
    begin
          if (ren==2'b10)
              begin
              rdata2 <= rf[raddr2][31:16];
              end
           if (ren==2'b01)
              begin
              rdata2 <= rf[raddr2][15:0];
              end
            if (ren==2'b11)
              begin
              rdata2 <= rf[raddr2];
            end
            else
              begin
              rdata2 <= 32'd0;
              end
     end
```

（2）regfile.display 文件中修改相应输入、led 接口、实例化等

```
//时钟与复位信号
input clk,
input resetn,     //后缀"n"代表低电平有效

//拨码开关，用于产生写使能和选择输入数
input [3:0]wen,
input [1:0]ren,
input [1:0] input_sel,
```

```
regfile rf_module(
    .clk   (clk   ),
    .wen   (wen   ),
    .ren(ren),
    .raddr1(raddr1),
    .raddr2(raddr2),
    .waddr (waddr ),
    .wdata (wdata ),
    .rdata1(rdata1),
    .rdata2(rdata2),
    .test_addr(test_addr),
    .test_data(test_data)
);
```

（3）refile.xdc 文件中修改对应开关接口以及删除多余 led 灯

拨码开关从左向右依次控制写入位置的 wen、控制当前操作方式的 input_select、控制读高低位的 ren。

```
#led 灯连接，用于输出

#set_property PACKAGE_PIN K23 [get_ports led_wen1]
set_property PACKAGE_PIN F7 [get_ports led_wen[0]]
set_property PACKAGE_PIN H7 [get_ports led_wen[1]]
set_property PACKAGE_PIN H8 [get_ports led_wen[2]]
set_property PACKAGE_PIN G8 [get_ports led_wen[3]]

set_property PACKAGE_PIN D5 [get_ports led_waddr]
set_property PACKAGE_PIN A4 [get_ports led_wdata]
set_property PACKAGE_PIN A3 [get_ports led_raddr1]
set_property PACKAGE_PIN A5 [get_ports led_raddr2]
#拨码开关连接，用于输入，依次为 sw0,sw1,sw7
set_property PACKAGE_PIN AC21 [get_ports wen[0]]
set_property PACKAGE_PIN AD24 [get_ports wen[1]]
set_property PACKAGE_PIN AC22 [get_ports wen[2]]
set_property PACKAGE_PIN AC23 [get_ports wen[3]]

set_property PACKAGE_PIN AA7 [get_ports input_sel[1]]
set_property PACKAGE_PIN Y6      [get_ports input_sel[0]]
set_property PACKAGE_PIN AB6    [get_ports ren[1]]
set_property PACKAGE_PIN W6 [get_ports ren[0]]

set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports resetn]
# set_property IOSTANDARD LVCMOS33 [get_ports led_wen]
```

5、

（1） 输入部分，向 1A 写入 0251125DD

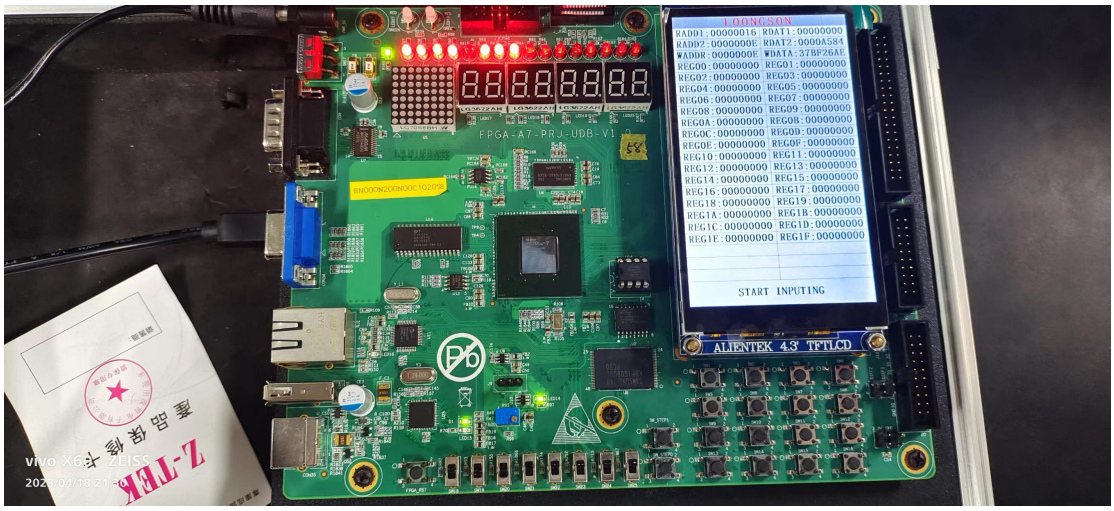
输入最高 2 位：将 wen 的四个开关置为 0（即前四个开关向上），input_sel 置 10（即开关 56 置下上），向 WADDR 写入 0E（即向 1A 写入数据），再将 56 置为 11，紧接着四次写入代写数据。

I. 写入 02，拨动开关 4 向下，将 AE 写入，如图：

II. 写入 11，拨动开关 3 向下，将 26 写入，如图：

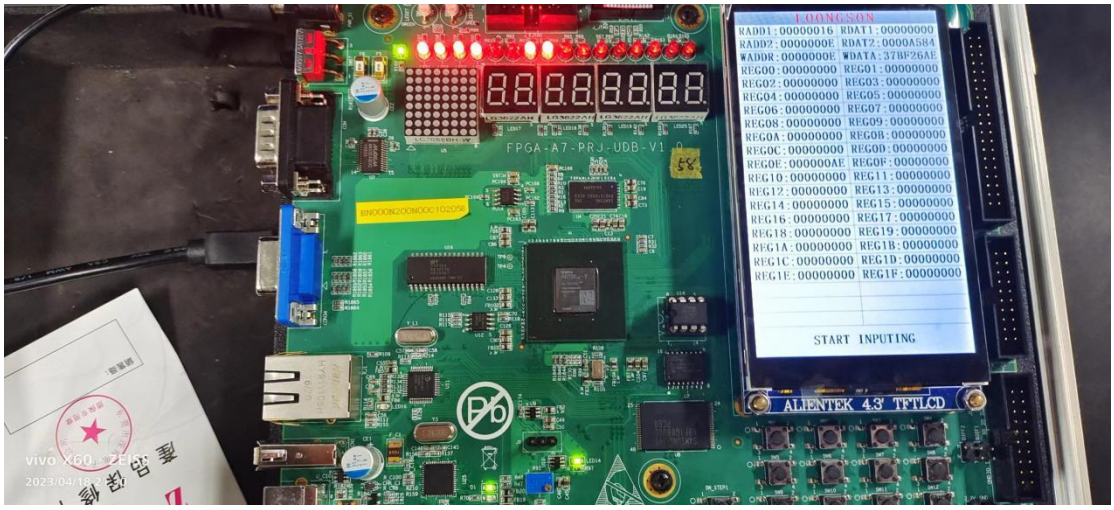
III. 写入 25，拨动开关 2 向下，将 BF 写入，如图：

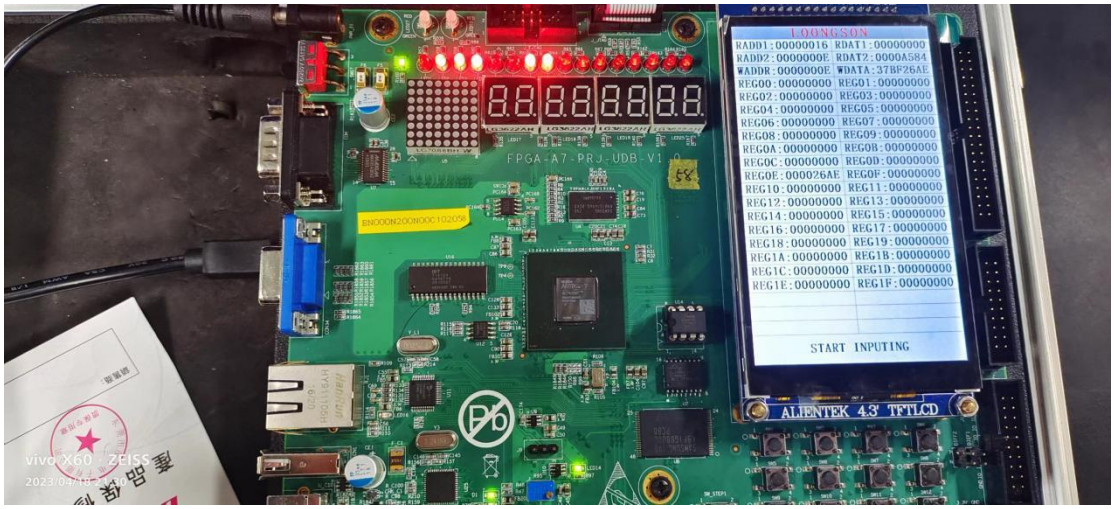
IV. 写入 DD，拨动开关 1 向下，将 37 写入，如图：



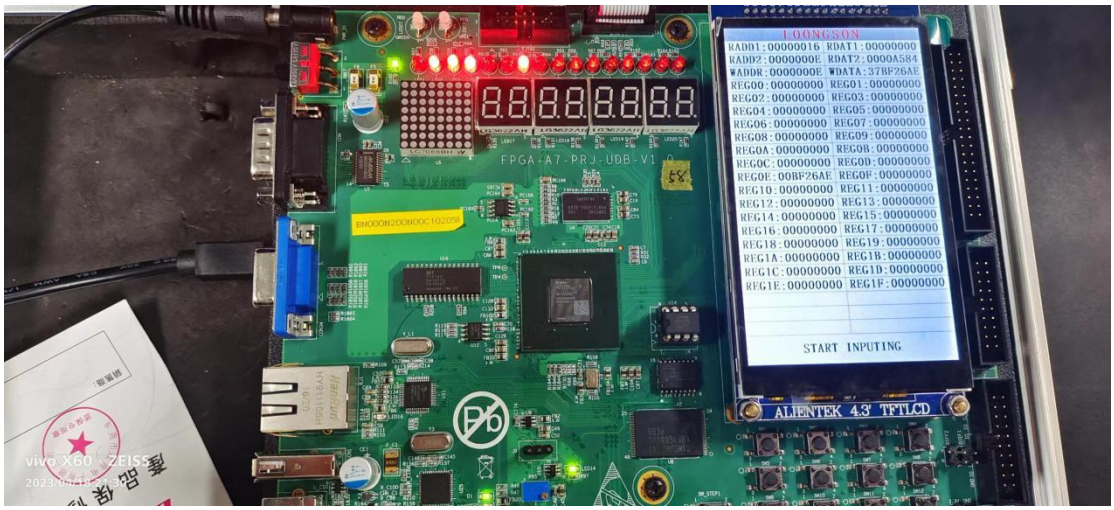
（2） 读数据

I. 读取 OE 的低 16 位，如图

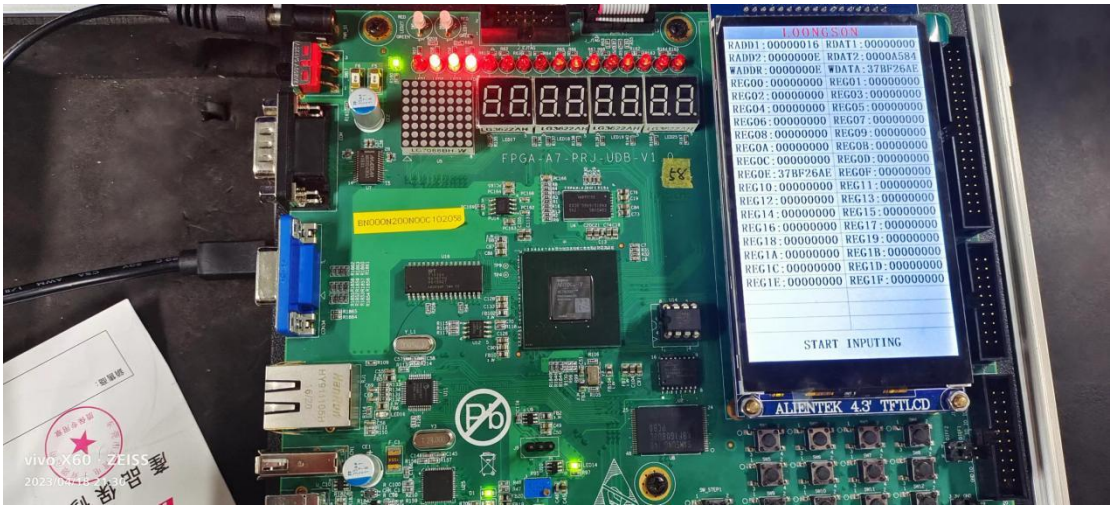




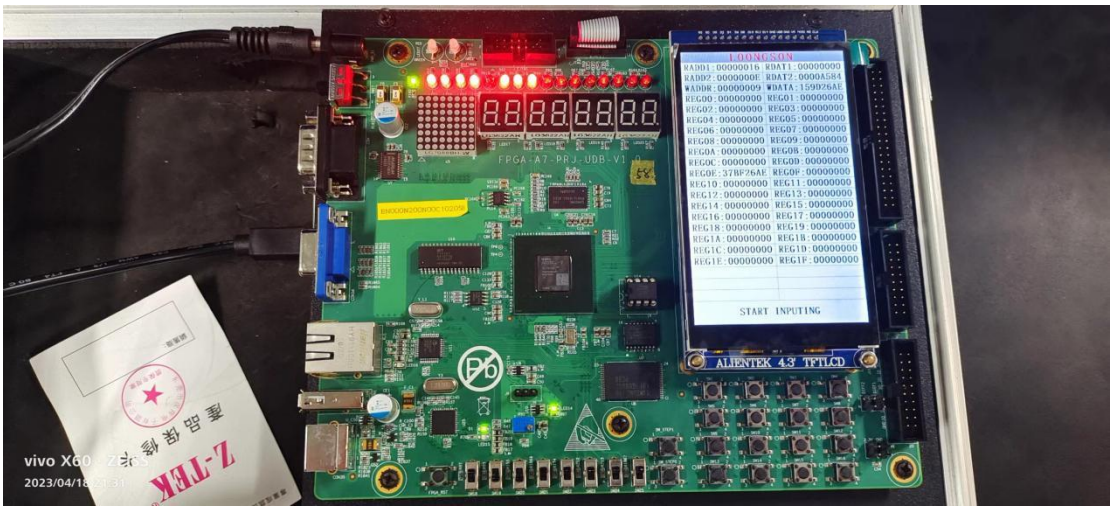
II. 读取 OE 的低 16 位，如图



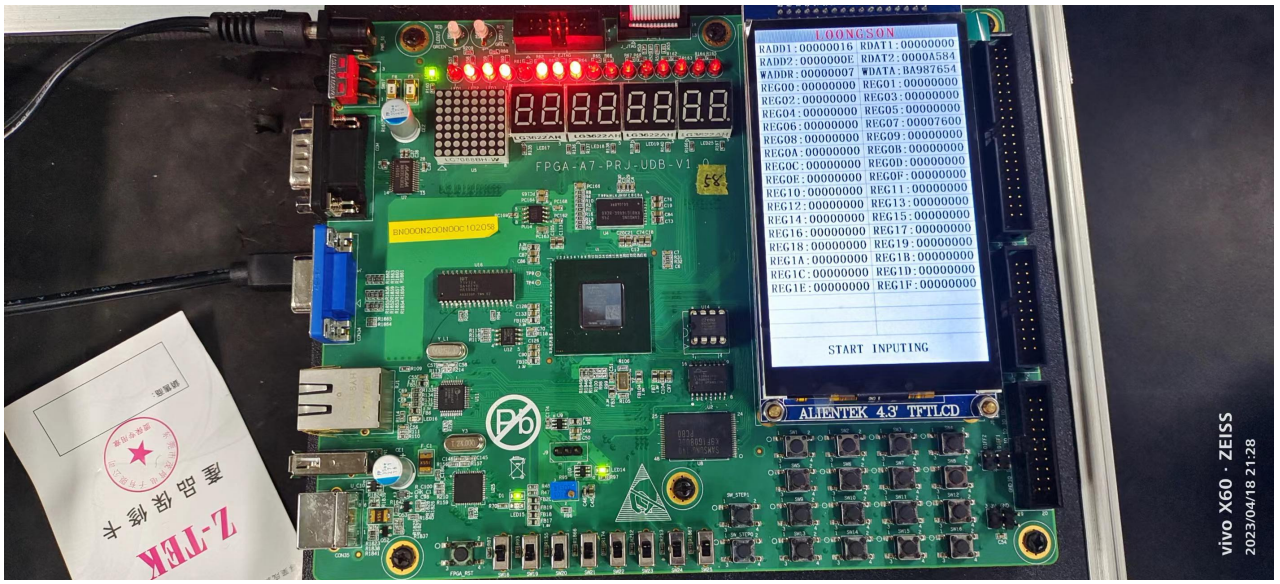
III. 读取 OE 的全 32 位，如图

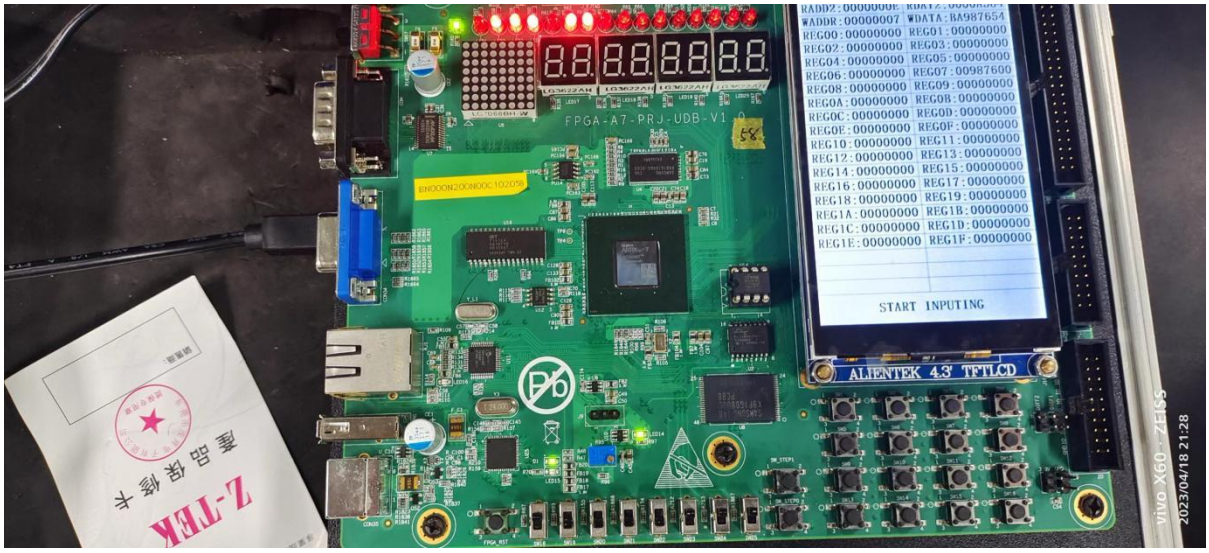


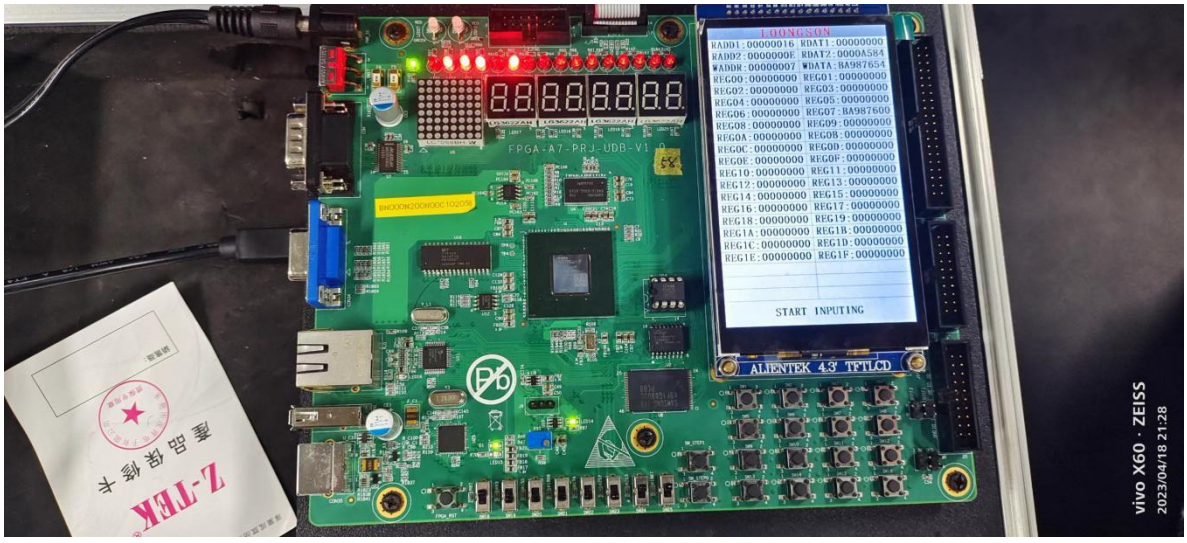
5.调整开关，也可以将内容先输入一个 RADT 当中，再进行下一步的输入

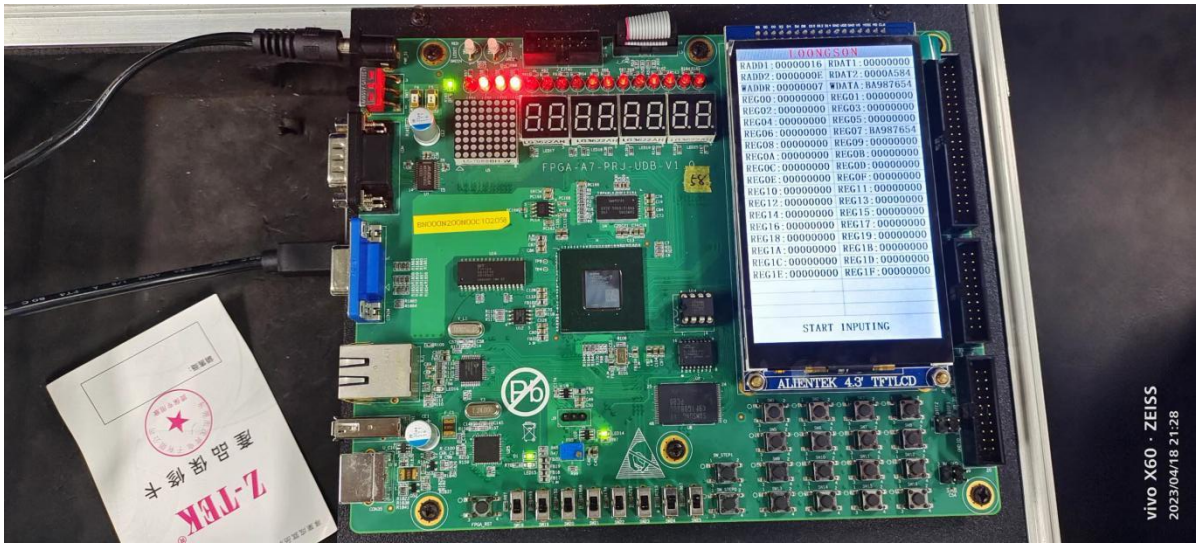


同样的方法我们可以进行不同位置不同顺序的输入









6、 总结感想

1. 对于理论课学习的寄存器堆的概念更加清晰，记忆更加深刻，通过本次实验，我们深入理解了寄存器堆的原理和实现方法，掌握了 Verilog 语言的基本语法和使用方法，提高了我们的电路设计和 Verilog 编程能力。此外在修改原代码使之可以按 4 个高低位置输入时，并在对 verilog 语言不熟悉的基础上，经历了诸多难题，对这门语言的运用更熟练

2. 对我动手操作实验提升许多，在开始得到代码上箱操作时，对本次实验的开关操作以及寄存器如何进行读写仍然缺少认知，对于一些复杂的操作和变换也是无从下手。所以先认真审计代码，读懂 inp_sel 的作用以及使用方法（00、01、10、11 分别读、写目的寄存器、写数据），读懂 wen 为 1 时改变寄存器的值…最后再加上修改后的开关和逻辑，共 8 开关，慢慢最后得到实验结果。